Mathematical Preliminaries for Lossless Compression

C.M. Liu
Perceptual Lab, College of Computer Science
National Chiao-Tung University

http://www.csie.nctu.edu.tw/~cmliu/Courses/Compression/

Office: EC538
(03)5731877
cmliu@cs.nctu.edu.tw
Outlines

- Introduction
- Information Theory
- Models
- Coding
Achieving Data Compression

- Most data has natural redundancy
  - I.e., ‘straightforward’ encoding contains more data than the actual information in the data
  - E.g., audio sampling:
Achieving Data Compression (2)

- Compression == ‘squeezing out’ the inefficiencies of the information representation
  - Note #1: in lossy compression we *threw out* less important/imperceptible information
  - Note #2: We must be able to reverse the process to make the data usable again

- Q1: What data can be compressed?
- Q2: By how much?
- Q3: How close are we to optimal compression?
- *Information Theory*: a mathematical description of information and its properties
Representing Data

- Analog (continuous) data
  - Represented by real numbers
  - Note: cannot be represented by computers

- Digital (discrete) data
  - Given a finite set of symbols \{a_1, a_2, \ldots, a_n\},
  - All data represented as symbol sequences (or strings) in the symbol set
  - E.g.: \{a,b,c,d,r\} => abc, car, bar, abracadabra, ...
  - We use digital data to approximate analog data
Common Symbol Sets

- Roman alphabet plus punctuation
- ASCII - 256 symbols
- Braille, Morse
- Binary - \{0,1\}
  - 0 and 1 are called bits
  - All digital data can be represented efficiently in binary
  - E.g.: \{a, b, c, d\} fixed length binary representation (2 bits/symbol):

<table>
<thead>
<tr>
<th>Symbol</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
Information

- First formally developed by Claude Shannon at Bell Labs in the 1940s/50s
- Explains limits on coding/communication using probability theory
- **Self-information**
  - Given event $A$ with probability $P(A)$

\[ i(A) = \log_b \frac{1}{P(A)} = -\log_b P(A) \]
Observations
- Low $P(A)$ => high $i(A)$
- High $P(A)$ => low $i(A)$

Rationale:
- Low probability (surprise) events carry more information; think man bites dog vs. dog bites man

Suppose $A$ and $B$ are independent then
- $i(AB) = i(A) + i(B)$

\[
 i(AB) = \log_b \frac{1}{P(AB)} = \log_b \frac{1}{P(A)P(B)} = \\
= \log_b \frac{1}{P(A)} + \log_b \frac{1}{P(B)} = i(A) + i(B) 
\]
Coin Flip Example

- **Fair coin**
  - Let H & T be the outcomes
  - If $P(H) = P(T) = 1/2$, then
  - $i(H) = i(T) = -1/\log_2(1/2) = 1$ bit

- **Unfair coin**
  - Let $P(H) = 1/8, P(T) = 7/8$
  - $i(H) = 3$ bits
  - $i(T) = 0.193$ bits

- Note that $P(H) + P(T) = 1$
(First-Order) Entropy

Let

- \( A_1, \ldots, A_n \) be all the independent possible outcomes from an experiment
- with probabilities \( P(A_1), \ldots, P(A_n) \)

\[
H = \sum_{i=1}^{n} P(A_i) i(A_i) = - \sum_{i=1}^{n} P(A_i) \log_b P(A_i)
\]

- If the experiment generates symbols, then (for \( b=2 \)) \( H \) is the average number of binary symbols needed to code the symbols.
- **Shannon**: No lossless compression algorithm can do better.
- **Note**: The general expression for \( H \) is more complex but reduces to the above for iid sources
Consider the sequence:

\[1\ 2\ 3\ 2\ 3\ 4\ 5\ 4\ 5\ 6\ 7\ 8\ 9\ 8\ 9\ 10\]

Assume it correctly describes the probabilities generated by the source; then

\[P(1) = P(6) = P(7) = p(10) = \frac{1}{16}\]
\[P(2) = P(3) = P(4) = P(5) = P(8) = P(9) = \frac{2}{16}\]

Assuming the sequence is iid

\[
H = - \sum_{i=1}^{10} P(i) \log_2 P(i) = -4 \frac{1}{16} \log_2 \left( \frac{1}{16} \right) - 6 \frac{2}{16} \log_2 \left( \frac{2}{16} \right) = 3.25\text{bits}
\]
Entropy Example #2

- Assume sample-to-sample correlation
- Instead of coding samples, code difference:
  - 1 1 1 -1 1 1 1 -1 1 1 1 1 1 -1 1 1
  - Now P(1) = 13/16, P(-1) = 3/16
  - H = 0.70 bits (per symbol)
- Model also needs to be coded

- Knowing something about the source can help us ‘reduce’ the entropy
  - Note the we cannot actually reduce the entropy of the source, as long as our coding is lossless
  - Instead, we are reducing our estimate of the entropy
Entropy Example #3

- Consider the sequence:
  - 1 2 1 2 3 3 3 3 3 3 1 2 3 3 3 3 1 2
  - \( P(1) = P(2) = \frac{1}{4}, \ P(3) = \frac{1}{2}, \ H = 1.5 \text{ bits/symbol} \)
  - Total bits: 20 x 1.5 = 30

- Reconsider the sequence
  - (1 2) (1 2) (3 3) (3 3) (1 2) (3 3) (3 3) (1 2) (3 3) (1 2)
  - \( P(1 2) = \frac{1}{2}, \ P(3 3) = \frac{1}{2} \)
  - \( H = 1 \text{ bit/symbol} \times 10 \text{ symbols} = 10 \text{ bits} \)

- In theory, structure can eventually be extracted by taking larger samples

- In reality, we need an accurate model as it is often impractical to observe a source for long
Models

- Physical models
  - Based on understanding of the process generating the data
    - E.g., speech
  - A good model leads to good compression
  - Usually impractical
  - Empirical data instead
    - Statistical methods can help take a proper sample
Probability Models

- **Ignorance model**
  1. Assume each letter is generated independently from the rest
  2. Assume all letters are generated with equal probability
  - Examples?
    - ASCII, RGB, CDDA, …

- **Improvement**—drop assumption 2:
  - $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, $\mathcal{P} = \{P(a_1), P(a_2), \ldots, P(a_n)\}$
  - Very efficient coding schemes exist already

- **Note**
  - If 1. does not hold, a better solution likely exists
Markov Models

- Assume that each output symbol depends on previous $k$ ones. Formally:
  - Let $\{x_n\}$ be a sequence of observations
  - We call $\{x_n\}$ a $k^{th}$-order discrete Markov chain (DMC) if
    \[
P(x_n | x_{n-1}, \ldots, x_{n-k}) = P(x_n | x_{n-1}, \ldots, x_{n-k}, \ldots)
    \]
  - Usually, we use a first-order DMC:
    \[
P(x_n | x_{n-1}) = P(x_n | x_{n-1}, \ldots, x_{n-k}, \ldots)
    \]

- Linear dependency model
  - $x_n = \rho x_{n-1} + \varepsilon_n$
  - $\varepsilon_n \Rightarrow$ white noise
Non-linear Markov Models

- Consider a BW image as a string of black & white pixels (e.g. row-by-row)
  - Define two states: $S_b$ & $S_w$ for the current pixel
  - Define probabilities:
    - $P(S_b) = \text{prob of being in } S_b$
    - $P(S_w) = \text{prob of being in } S_w$
  - Transition probabilities
    - $P(b|b), P(b|w)$
    - $P(w|b), P(w|w)$

\[
\begin{align*}
H(S_w) &= -P(b/w)\log(b/w) - P(w/w)\log(w/w) \\
H(S_b) &= -P(w/b)\log(b/w) - P(b/b)\log(b/b) \\
P(w/w) &= 1 - P(b/w), \quad P(b/b) = 1 - P(w/b) \\
H &= P(S_b)H(S_b) + P(S_w)H(S_w)
\end{align*}
\]
Markov Model (MM) Example

Assume

\[
P(S_w) = \frac{30}{31} \quad P(S_b) = \frac{1}{31}
\]

\[
P(w/w) = 0.99 \quad P(b/w) = 0.01 \quad P(b/b) = 0.7 \quad P(w/b) = 0.3
\]

- For the \textit{iid} model:

\[
H_{iid} = -0.8 \log 0.8 - 0.2 \log 0.2 = 0.206
\]

- For the \textit{Markov} model:

\[
H(S_b) = -0.3 \log 0.3 - 0.7 \log 0.7 = 0.881
\]

\[
H(S_w) = -0.01 \log 0.01 - 0.99 \log 0.99 = 0.081
\]

\[
H_{Markov} = \frac{30}{31} \cdot 0.081 + \frac{1}{31} \cdot 0.881 = 0.107
\]
In written English, probability of next letter is heavily influenced by previous ones
- E.g. u after q

Shannon’s work
- 2nd-order MM, 26 letters + space $H = 3.1$ bits/letter
- Word-based model $H=2.4$ bits/letter
- Human prediction based on 100 letters $0.6 \leq H \leq 1.3$ bits/letter

Longer context => better prediction

Practical concerns:
- Context model storage (e.g. 4th-order w/ 95 chars = $95^4$ contexts)
- Zero frequency problem
Many sources cannot be adequately described by a single model

E.g.: an executable contains:
- Code, resources (text, images, …)

Solution: composite model:
Coding

- **Alphabet**
  - Collection of symbols called letters

- **Code**
  - A set of binary sequences called codewords

- **Coding**
  - The process of mapping letters to codewords
  - Fixed vs. variable-length coding

- **Example: letter ‘A’**
  - ASCII: 01000001
  - Morse: •—

- **Code rate**
  - *Average* number of bits per symbol
Example

- Alphabet = \{a_1, a_2, a_3, a_4\}
- P(a_1) = 1/2, P(a_2) = 1/4, P(a_3) = P(a_4) = 1/8
- H = 1.75 bits
- n(a_i) = length (codeword(a_i)), i=1..4
- Avg length \( l = \sum_{i=1..4} P(a_i) n(a_i) \)

Possible codes:

<table>
<thead>
<tr>
<th></th>
<th>Probability</th>
<th>Code 1</th>
<th>Code 2</th>
<th>Code 3</th>
<th>Code 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1</td>
<td>0.500</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a_2</td>
<td>0.250</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>a_3</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>a_4</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
<tr>
<td>(l)</td>
<td>1.125</td>
<td>1.250</td>
<td>1.750</td>
<td>1.875</td>
<td></td>
</tr>
</tbody>
</table>
Uniquely Decodable Codes (2)

<table>
<thead>
<tr>
<th></th>
<th>Probability</th>
<th>Code 1</th>
<th>Code 2</th>
<th>Code 3</th>
<th>Code 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>0.500</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$a_2$</td>
<td>0.250</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.125</td>
<td>1</td>
<td>00</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.125</td>
<td>10</td>
<td>11</td>
<td>111</td>
<td>0111</td>
</tr>
<tr>
<td>$l$</td>
<td>1.125</td>
<td>1.250</td>
<td>1.750</td>
<td>1.875</td>
<td></td>
</tr>
</tbody>
</table>

- **Code 1**
  - Identical codewords for $a_1$ & $a_2$: decode(‘00’) = ?

- **Code 2**
  - Unique codes but ambiguous: decode(‘00’/‘11’) = ?

- **Code 3**
  - Uniquely decodable, instantaneous

- **Code 4**
  - Uniquely decodable, ‘near-instantaneous’
Unique decodability:

- Given any sequence of codewords, there is a unique decoding of it.

Unique != instantaneous

E.g.:

- $a_1 \leftrightarrow 0$
- $a_2 \leftrightarrow 01$
- $a_3 \leftrightarrow 11$

decode(0111111111) = $a_1a_3...$ or $a_2a_3...$?

- don’t know until the end of the string

0111111111 $\rightarrow$ 01111111$a_3$ $\rightarrow$ 011111$a_3a_3$ $\rightarrow$ 0111$a_3a_3a_3$ $\rightarrow$

01$a_3a_3a_3a_3$ $\rightarrow$ $a_2a_3a_3a_3a_3a_3$
Unique Decodability Test

- Prefix & dangling suffix
  - Let \( a = a_1 \ldots a_k, b = b_1 \ldots b_n \) be binary codewords and \( k < n \)
  - If \( a_1 \ldots a_k = b_1 \ldots b_k \) then \( a \) is a prefix of \( b \) and
  - \( b_{k+1} \ldots b_n \) is a dangling suffix: \( ds(a, b) \)

- Algorithm
  - Let \( C = \{c_n\} \) be the set of all codewords
    - For all pairs \((c_i, c_j)\) in \( C \) repeat
      - If \( ds(c_i, c_j) \notin C \) \(//\) dangling suffix is not a codeword
        - \( C_I = C_I \cup ds(c_i, c_j) \)
      - Else \(//\) dangling suffix is a codeword
        - return NOT_UNIQUE
    - until no more unique pairs
  - return UNIQUE
Prefix Codes

- **Prefix code:**
  - No codeword is prefix of another.
  - Prefix codes are also known as prefix-free codes, prefix condition codes, comma-free codes (although this is incorrect), and instantaneous codes.

- **Binary trees as prefix decoders:**

```
repeat
  curr = root
  repeat
    if get_bit(input) = 1
      curr = curr.right
    else
      curr = curr.left
  until is_leaf(curr)
output curr.symbol
until eof(input)
```
Decoding Prefix Codes: Example

<table>
<thead>
<tr>
<th>symbol</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
</tr>
<tr>
<td>c</td>
<td>110</td>
</tr>
<tr>
<td>d</td>
<td>1110</td>
</tr>
<tr>
<td>r</td>
<td>1111</td>
</tr>
</tbody>
</table>

abracadabra = 0101110110011100101110
Decoding Example

Input = 010111101100111001011110

Output = -----------
Decoding Example

Input  = 01011101110011100101110
Output = a----------
Decoding Example

Input  = $-10111101100111001011110$

Output  = a----------
Decoding Example

Input  = 0111101100111001011110
Output = a----------
Decoding Example

Input  = --0111101100111001011110
Output = ab-----------
Decoding Example

Input = 111101100111001011110
Output = ab---------
Decoding Example

Input  = ---11101100111001011110
Output = ab---------
Decoding Example

Input = ----11101100111001011110
Output = ab---------
Decoding Example

Input = ------1101100111001011110
Output = ab--------
Decoding Example

Input = --------101100111001011110
Output = abr--------
Decoding Example

Input = --------01100111001011110
Output = abr--------
Decoding Example

Input = --------01100111001011110
Output = abra--------
Decoding Example

Input  = ---------1100111001011110
Output = abra-------
Decoding Example

Input = --------1100111001011110
Output = abra--------
Decoding Example

Input = ------------100111001011110
Output = abra-------
Decoding Example

Input  = -------------0111001011110
Output = abraç-------
Decoding Example

Input  = ------------011001011110
Output = abrac------
Decoding Example

Input  = -----------0111001011110
Output = abracadaba------
Decoding Example

Input = ------------111001011110
Output = abraca-----

and so on …
Summary

- Basic definitions of Information Theory
  - Information
  - Entropy
  - Models
  - Codes
    - Unique decodability
    - Prefix codes
- Homeworks (pp. 38-39)
  - 3, 4, 7.
- Program (pp. 39)
  - 5